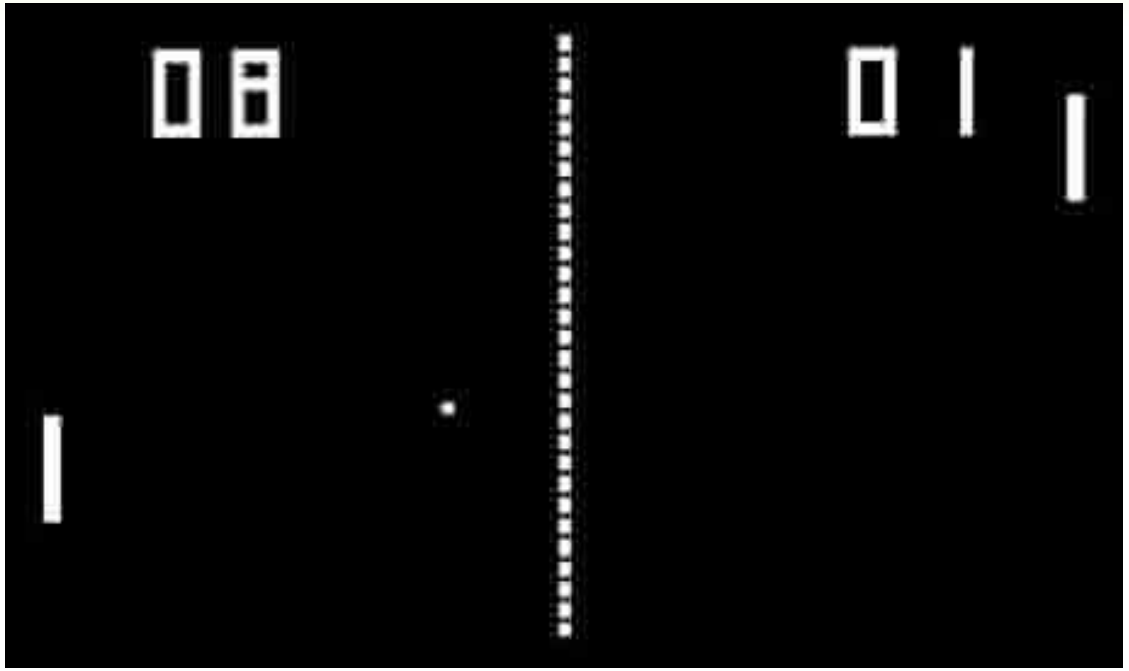
Parallelisation in Game Engines

By: Daniel Kerr

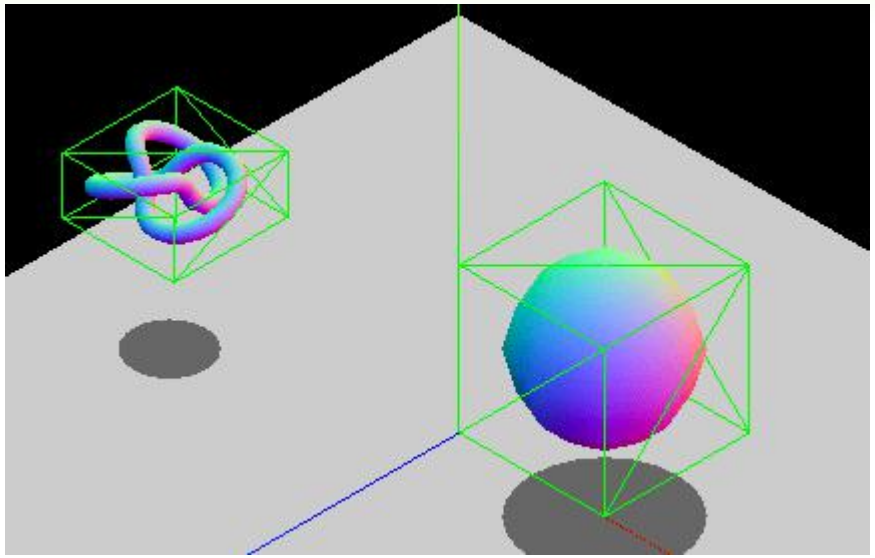
Why do we need parrallisation?

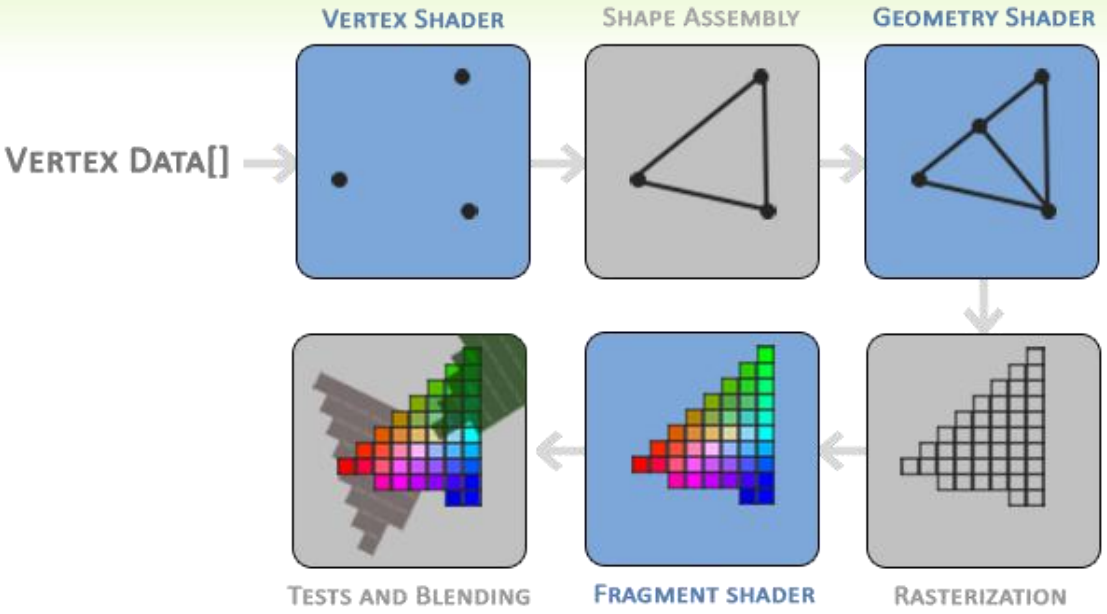
- Early games didn’t need it
- Modern games are more intensive
- 60Hz - 144Hz refresh rates
- Optimizing performance



Parrallism Types

- Task Parallism
  - Different things at once
  - Moving with Collision Detection
- Data Parallism
  - One thing lots of times
  - Using Vertex Shaders





Game Loop

- Loop that handles gameplay
- Handling HIDs
- Update game states
- Calls render jobs

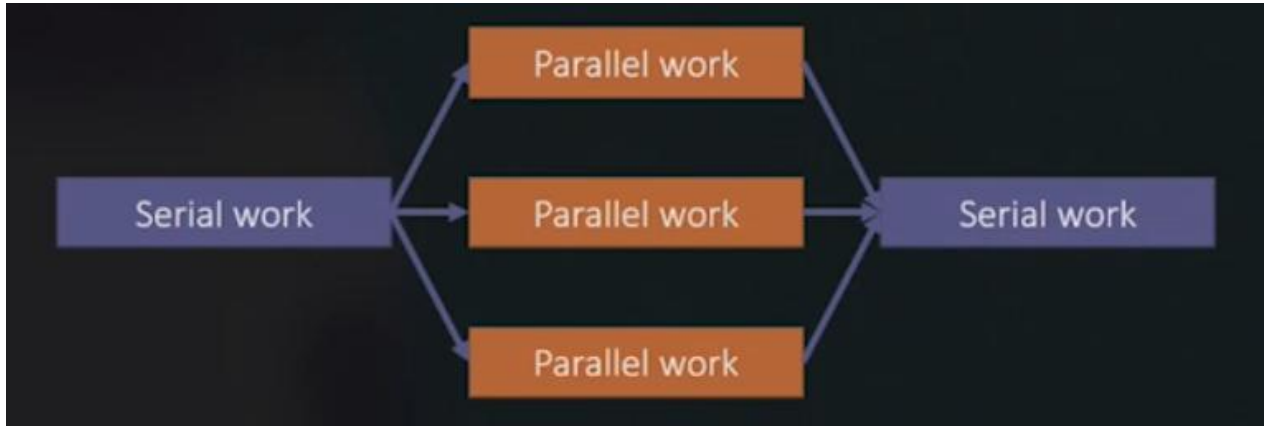
```
void main() {
    init();                 // Initialize everything before starting the game
    while (true){           // Game Loop
        readHumanInterfaceDevice();
        if (quitRequest()) {
            break;          // Exit the Game Loop
        }
        pollEvents();       // Group relevent data
        update();           // Update the Game State
        handleEvents();     // Handle collisions, physics, etc.
        render();           // Call Render Job from GPU
    }

}
```

Parrallelisation Options

- One Thread per Subsystem
  - Main thread spawns subsystems
  - Not scalable with hardware
  - Waisted compute time

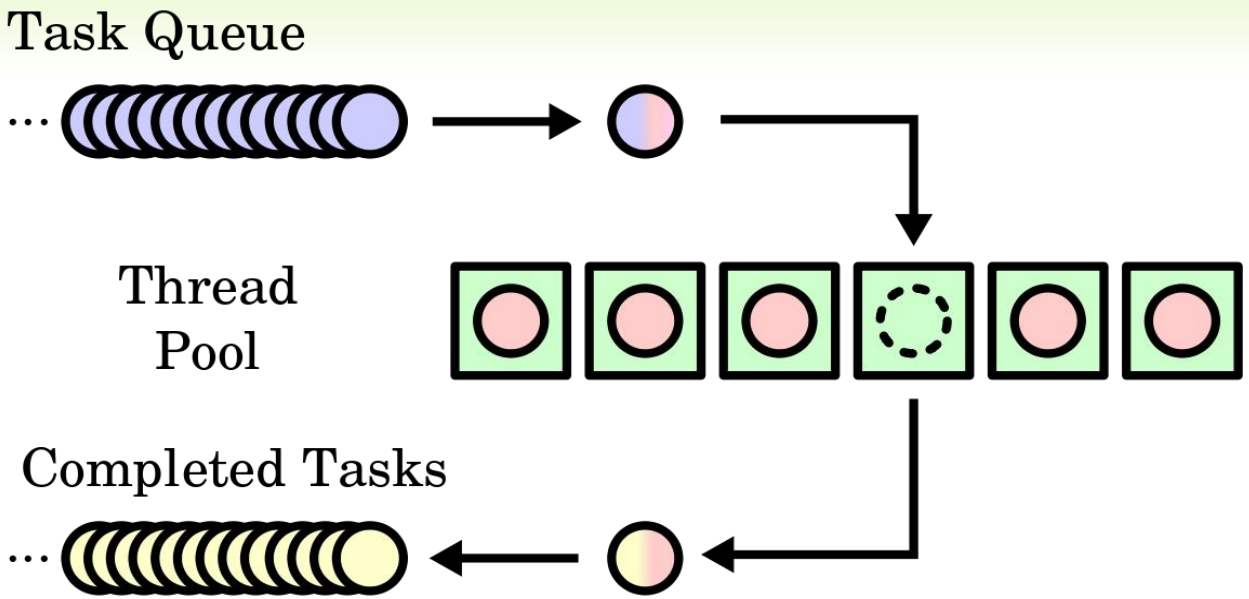
Parrallelisation Options

- Fork / Join
  - Next logical step
  - Spawn threads as needed
  - Expensive
- Optimization
  - SIMD design
  - Vectorize loops
  - Thread Pools



Thread Pools

- Spawn threads at start
- Scaleable to hardware
- Difficult to maintain



Parrallelisation Options

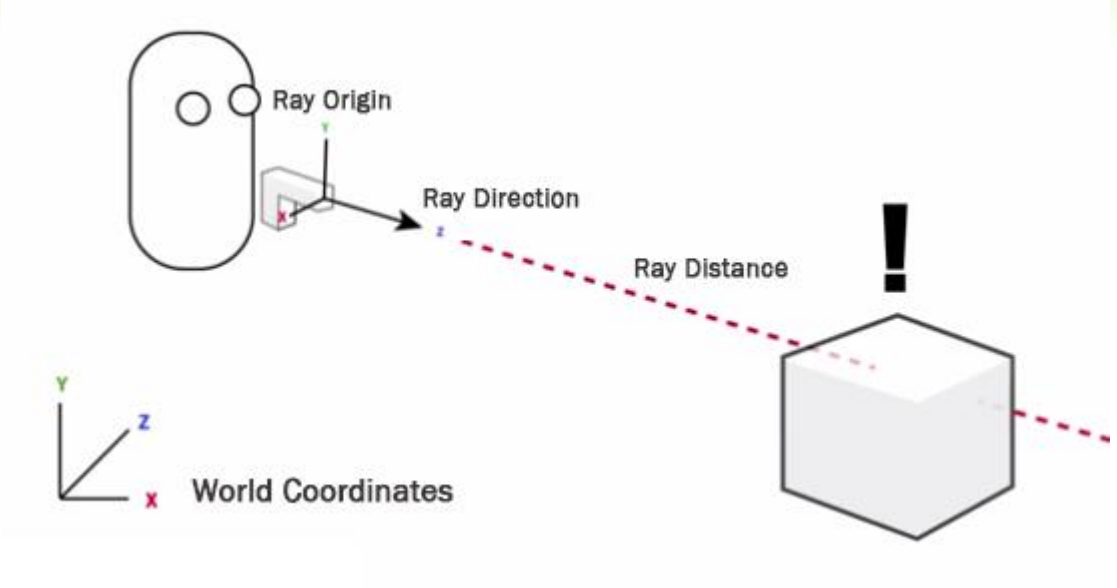
- Job System
  - Queue datastructure
  - Jobs work independantly
  - Arbitrarily fine-graned
  - Highly scalable

Implementing Jobs

- Methods
  - Kick
  - Wait
  - Sleep
  - Wait
- Priority
- Locks

Sleeping Jobs

- Cannot sleep with simple job implementation
- Have to fully context switch

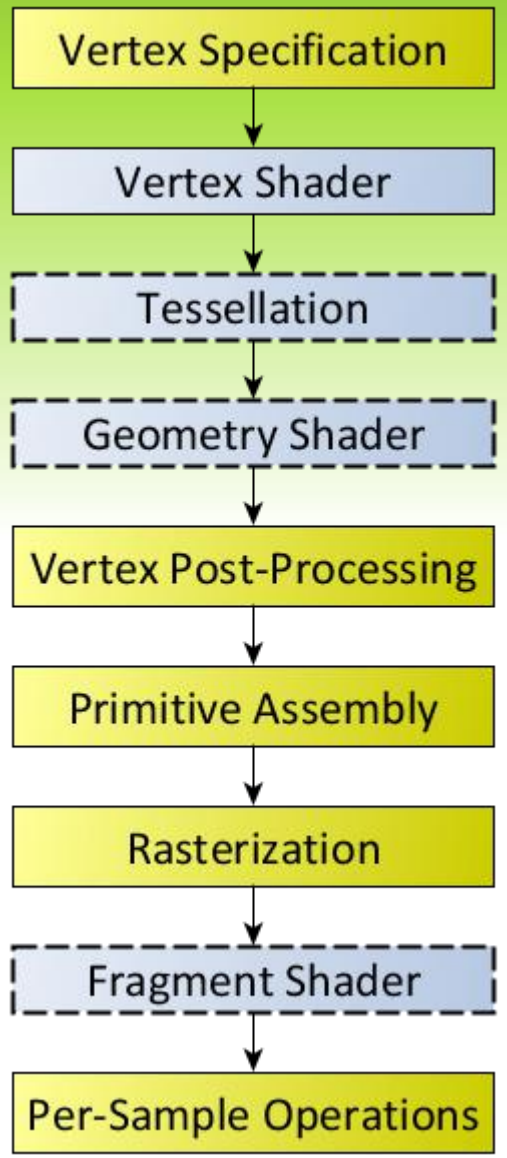


More Job Implementations

- Coroutines
  - Yield to another coroutine with explicit call
- Job Counters
  - Track jobs running
  - Essentially Fork / Join
  - Good for batch jobs
- Fibers
  - Widely used
  - Never scheduled by kernel
  - Cooperatively scheduled by members

Working with the GPU

- Shaders
- GLSL
- Send manageable amounts of data
- Keep O(1) execution time



End

Thank you for watching